CLAIMS

1. A system for operating automatically on source code (17) submitted by users to generate optimized code (19) suitable for running on a predefined hardware platform (90) comprising at least one processor (91), and for use in a predetermined field of application, the system being characterized in that it comprises means (51, 52) for receiving symbolic code sequences referred to as benchmark sequences (1) representative of the behavior of said processor (91) in terms of performance, for the predetermined field of application; means (53) for receiving first static parameters (2) defined on the basis of the predefined hardware platform (90), its processor (91), and the benchmark sequences (1); means (55) for receiving dynamic parameters (7) also defined on the basis of the predefined hardware platform (90), its processor (91), and the benchmark sequences (1); an analyzer device (10) for establishing optimization rules (9) from tests and measurements of performance carried out using the benchmark sequences (1), the static parameters (2), and the dynamic parameters (7); a device (80) for optimizing and generating code receiving firstly the benchmark sequences (1) and secondly the optimization rules (9) for examining the user source code (17), detecting optimizable loops, decomposing them into kernels, and assembling and injecting code to deliver the optimized code (19); and means (74) for reinjecting information coming from the device (80) for

generating and optimizing code and relating to the kernels back into the benchmark sequences (1).

2. A system according to claim 1, characterized in that the analyzer device (10) comprises a test generator (3) connected firstly to the means (51) for receiving benchmark sequences and secondly to the means (53) for receiving static parameters in order to generate automatically a large number of test variants which are transferred by transfer means (61) to be stored in a variant database (4); an exerciser (5) connected firstly to transfer means (62) for receiving the test variants stored in the variant database (4) and secondly to the means (55) for receiving dynamic parameters to execute the test variants in a range of variation of the dynamic parameters (7) and produce results which are transferred by transfer means (63) for storage in a results database (6); and an analyzer (8) connected to the transfer means (64) to receive the results stored in the results database (6), to analyze them, and to deduce therefrom optimization rules which are transferred by transfer means (57) into an optimization rules database (9).

3. A system according to claim 2, characterized in that the analyzer (8) includes filter means having an arbitrary threshold for optimum performance, so as to consider a variant in the results database as being optimal in the parameter space providing it satisfies the filter criteria.

4. A system according to claim 2 or claim 3, characterized in that the analyzer (8) further comprises means (54) for modifying the static parameters (2) and means (56) for modifying the dynamic parameters (7).

5. A system according to any one of claims 1 to 4, characterized in that the device (80) for optimizing and generating code comprises a device (18) for generating optimized code and an optimizer (12), the optimizer comprising a strategy selection module (13) connected firstly to the means (92) for receiving kernels identified in the original source code, secondly to the means (52) for receiving benchmark sequences (1), and thirdly to means (58) for receiving optimization rules (9) so as to generate, for each kernel corresponding to a tested benchmark sequence, a plurality of versions (15), each being optimal under a certain combination of parameters, and a combination and assembly module (14) connected to the means (59) for receiving optimization rules (9), to means (66) for receiving information coming from the strategy selection module (13), and to means (68) for receiving the plurality of versions (15), in order to deliver via transfer means (93) information comprising the corresponding optimized versions, their utilization zone, and where appropriate the test to be executed in order to determine dynamically which version is the most suitable.

6. A system according to claim 5, characterized in that it comprises an optimized kernel database (16), and in that the combination and assembly module (14) is connected to the optimized kernel database (16) by transfer means (79) for storing information in said optimized kernel database, said information comprising the optimized versions, their utilization zones, and where appropriate the test to be executed in order to determine dynamically which version is the most suitable.

7. A system according to any one of claims 1 to 6, characterized in that the device (80) for optimizing and generating code comprises an optimizer (12) and a device (18) for generating optimized code, which device comprises a module (20) for detecting optimizable loops that is connected to means (71) for receiving user source code (17), a module (22) for decomposing them into kernels, a module (23) for case analysis, assembly, and code injection that is connected via transfer means (92) to the optimizer (12) to transmit the identity of the detected kernel, and transfer means (93) for receiving the information describing the corresponding optimized kernel, with the module (23) for case analysis, assembly, and code injection also being connected to means (73) for supplying optimized code.

8. A system according to claims 6 and 7, characterized in that the module (23) for case analysis, assembly, and code injection is also connected to the optimized kernel database (16) to

receive information describing an optimized kernel without invoking the optimizer (12) if the looked-for kernel has been stored therein.

9. A system according to claim 8, characterized in that the module (23) for case analysis, assembly, and code injection further comprises means (74) for adding to the benchmark sequences (1), kernels that have been discovered in the module (23) for case analysis, assembly, and code injection, without having the corresponding identity in the optimized kernel database (16) nor in the benchmark sequences.

10. A system according to any one of claims 6, 8, and 9, characterized in that it includes a compiler (81) and a link editor (82) for receiving reorganized source code (19) from the device (80) for optimizing and generating code, and for producing optimized binary code (83) adapted to the hardware platform (90).

11. A system according to claim 10, characterized in that it includes means (85) for transferring the source code for the optimized kernels from the optimized kernel database (16) to the compiler (81).

12. A system according to claim 10, characterized in that it includes a compiler (181) and an installation module (182) for

installing a dynamic library on the hardware platform (90), which library contains all of the capabilities of the optimized kernels.

13. A system according to any one of claims 1 to 12, characterized in that the predetermined field of application is scientific computation.

14. A system according to any one of claims 1 to 12, characterized in that the predetermined field of application is signal processing.

15. A system according to any one of claims 1 to 12, characterized in that the predetermined field of application is graphics processing.

16. A system according to any one of claims 1 to 15, characterized in that the benchmark sequences (1) comprise a set of simple and generic loop type code fragments specified in a source type language and organized in a hierarchy of levels by order of increasing complexity of the code for the loop body.

17. A system according to claim 16, characterized in that the benchmark sequences (1) comprise benchmark sequences of level 0 in which only one individual operation is tested and

corresponding to a loop body constituted by a single arithmetic expression represented by a tree of height 0.

18. A system according to claim 17, characterized in that the benchmark sequences comprise benchmark sequences of level 1 for which there are considered and tested: combinations of two level 0 operations; and level 1 benchmark sequence operations corresponding to loop bodies constituted either by a single arithmetic expression represented by a tree of height 1, or by two arithmetic expressions, each being represented by a tree of height 0.

19. A system according to claim 18, characterized in that the benchmark sequences (1) comprise benchmark sequences of level 1, for which there are considered and tested combinations of two level 1 operations or three level 0 operations.

20. A system according to any one of claims 16 to 19, characterized in that the static parameters (2) comprise in particular the number of loop iterations for each benchmark sequence, the table access step size and the type of operands, the type of instructions used, the preloading strategies, and the strategies for ordering instructions and iterations.

21. A system according to any one of claims 16 to 20, characterized in that the dynamic parameters (7) comprise in

particular the location of table operands in the various levels of the memory hierarchy, the relative positions of the table starting addresses, and the branching history.

22. A system according to any one of claims 6, 8, and 9, characterized in that the optimized kernel database (16) includes loop type source code sequences corresponding to code fragments that are real and useful and organized in levels in order of increasing complexity.

23. A system according to any one of claims 1 to 12, characterized in that the predefined hardware platform (90) comprises at least one Itanium type processor.

24. A system according to any one of claims 1 to 12, characterized in that the predefined hardware platform (90) comprises at least one Power or Power PC type processor.

25. A system according to any one of claims 13 to 15 and according to claim 23, characterized in that the optimization rules (9) comprise at least some of the following rules:

a) minimizing the number of writes, in the event of write performance that is poor compared with read performance;

b) the importance of using loading pairs in floating point;

c) adjusting the degree to which a loop is unrolled as a function of the complexity of the loop body;

d) using operational latencies of arithmetic operations;

e) applying masking techniques for short vectors;

f) using locality suffixes for memory accesses (reading, writing, preloading);

g) defining preloading distances;

h) performing degree 4 vectorization so as to avoid some of the L2 bank conflicts;

i) taking account of multiple variants in order to avoid other L2 bank conflicts and conflicts in the read/write queue;

j) taking account of performance improvements associated with different optimizations;

k) using branching chains that minimize wrong predictions (short vectors);

l) merging memory accesses (grouping pixels together); and

m) vectorizing processing on pixels.

26. A system according to any one of claims 13 to 15 and according to claim 24, characterized in that the optimization rules (9) comprise at least some of the following rules:

a) re-ordering reads in order to group cache defects together;

b) using preloading solely for writes;

c) adjusting the degree to which loops are unrolled as a function of the complexity of the loop body;

d) using operational latencies in arithmetic operations;

e) using locality suffixes for memory accesses (reading, writing, preloading);

f) defining preloading distances;

g) taking account of multiple variants to avoid conflicts in read/write queues; and

h) taking account of performance improvements associated with different optimizations.